Paricle Filter based SLAM and Textured Map Generation

1st Hao Xiang
department of Electrical and Computer Engineering
UC San Diego
haxiang@ucsd.edu

***Abstract*—simultaneous localization and mapping (SLAM) is a fundamental problem in the robitics field and has a wide range of applications in the High Definition Map and autonomous driving domain. In this report, I implemented the Particle Filter based SLAM and utilized the 2D image and 3D space association to create the texture map which could provide detailed texture and color information in the occupancy grid. Also, in this report, I will introduce the algorithms and techniques I used in order to make the particle filter more stable and accurate.**

***Index Terms*—SLAM, Particle Filter, Texture Map**

I. Introduction

SLAM aims to build a map from an unknown environment while keeping the trajectory of the agents. Leonard, John etc. [2] first introducted the concept of SLAM. Since then, SLAM has gradually become a key problem in the mobile robotics and has developed several approaches including Graph-SLAM [4], EKF-SLAM [1], Fast-SLAM [3] etc. The defficulty of the SLAM problem comes from the chicken-or-egg situation, i.e. the environment and the location both are unknown while estimating one needs the information of the other unknown one.

SLAM is very useful in many domain like outdoor/indoor robot navigation, building local map/global map as well as terrian mapping in the space. And with the development of deep learning and semantic segmentation, there is also new research on semantic slam.

In this project, I use partilce filter based approach to do the SLAM. Use particles as the state proposal and each maintain a weight which is the confidence for that state given the privous information. Use Filter techniques – prediction and update, I can get better locations of the particles, thus better estimation. Meanwhile, I use the particle with the highest confidence as current state and transform the lidar to head frame, then to body frame, then to wolrd frame by using the assumption that the current sate of the robot is the particle with the highest confidence. Then I update the map based on lidar scan via log-odds. If the log-odds of the cell is greater than some threshold, then predict the cell as occupied, otherwise free. After updating the map, we can use the updated map for the update step in the next iteration. Also I implement the texture map by utilizing the kinect camera. Associate the depth image and rgb color and then associate them with the map. I will discuss the technical details in the technical approach section.

II. Problem Formulation

SLAM: Given the lidar scan data $z_{1:T}$, head angles , relative odometry $u_{1:T}$ between last reading, we want to estimate the robot's pose $x_{1:T}$ and occupancy grid map m such that $x_t, m = \arg\max_{x_t,m} p(x_{1:t}, m|z_{1:t}, u_{1:t})$.

Here the substript t is the information for current time (Here my notation is different from the slides). Like u_t is the transformation between time $t-1$ and t and z_t is the observation at current state x_t. This problem can be subdevided to Localization and Mapping problem.

Localization: Assume the map m is known, given the observation $z_{1:l}$ and relative odometry $u_{1:k}$ as the motion input where $l \leq t$ and $k \leq t$, we want to estimate the x_t s.t. $x_t = \arg\max_{x_t} p(x_t|z_{1:l}, u_{1:k})$.

As a special case, when $l = t$ and $k = t$, then it is a update step. When $l = t-1$ and $k = t$, it is a prediction step.

Mapping: Assume the state $x_{1:t}$ is known, we want to estimate the m s.t. $m = \arg\max_m p(m|z_{1:t}, x_{1:t})$. Assume the cell is independent of each other, then we need to estimate $p(m|z_{1:t}, x_{1:t}) = \prod_i p(m_i|z_{1:t}, x_{1:t})$.

Textured Map: Assume we know the map m, the trajectory of the robots $x_{1:T}$, the head angles over time $\texttt{head angles}_{1:T}$, as well as the depth image $\texttt{depth}_{1:T}$ and rgb image $\texttt{rgb}_{1:T}$, we want to project the colored ground points onto the occupancy grids.

III. Technical approach

SLAM problem contains pose estimation and mapping. For the lidar, I will do a ground removal and restrict the distance of the lidar data. Before doing prediction and update, I choose to use the first lidar scan to build an initial map. For pose estimation, I use particle filter and it consists of prediction step (motion model) and update step (observation model). After that, I can get a better estimation of the robot's pose and use it to update the map. In this section I will describe the detail steps for above steps and discuss the techniques I used for better performance. After that, I will discuss the approach for texture mapping.

A. *Data Processing*

This section is implemented in the `dataloader.py` file.

1) **Data Synchronization** For each Lidar data, I search the nearset Joint data that is ahead of Lidar data in the time. And use two pointers ($indx1$ for lidar data and $indx2$

for joint data) to keep track of the index of two data. Initialize two pointers to zero. And start search from $indx2$ to the end of the joint data unitil it hits the first data that is ahead of current lidar data. Update $indx1$ and repeat this process for the next iteration. In such a way, the data synchronization is $O(N)$ which faster than just using $np.argmax$ each time, which takes $O(N^2)$ overall.

2) **Downsampling** In order to make the codes run faster and easier to debug, I implement a downsampling approach so the dataloader will load data in specified step length. I will calculate cumulative for the delta pose before the iteration begins. And for each iteration, I will output the delta pose between current time t and time $t - step$ by using the difference beween the cumulative sum. In such a way, I can achive $O(N)$ time complexity for iterating all the data. After experiments, I find that sometimes the downsampling result would be even better than the original full frame. This may due to the noise of the lidar points as well as the drifts of the pose estimation.
3) **Lidar data processing and ground removal** Since the lidar data has the maximum distance, and some points that are too near the robot may be the points that hit the rotbo's body. Thus I restrict the distance of the lidar data to $0.1m$ to $30m$. Also when the robot move its head, the lidar point may hit the ground. Thus it is needed to do a ground removal. A simple approach would be after transforming the lidar points to the world frame, I will remove the points that has the z-axis less than $0.1m$. For the computation efficiency, I choose to do the ground removal in the body frame. I will remove the points that has the z-axis less than $0.1 - 0.93 = -0.83m$ (0.93 is the height of the body frame).

B. *Particle Filter*

Particle FIlter uses delta functions (AKA particles $\mu_{t|t}^{(k)}, k = 1 \dots N$) to represent the probability $p_{t|t}$ and $p_{t+1|t}$ with the weights $\alpha_{t|t}^{(k)}$ representing the probability (AKA confidence) of the particle. By using sets of delta functions, we can avoid the computation difficulty of intergral and we just need to update the weights and positions of each particles. And it would also output the corresponding lidar and head angle data and increase the index by the step size.

1) **Prediction Step** For prediction step, I use the motion model to all the particles with noise generated from Gaussian noise. The formula is shown in equation.1.

$$\mu_{t+1|t}^{(k)} \leftarrow \mu_{t|t}^{(k)} + u_t + noise \tag{1}$$

Here u_t is the relative odometry in current time stamp. For the noise of x and y position, I set the standard deviation proportional to the magnitude of the motion inputs (delta pose in this project) and set the mean to zero. For yaw, I set the mean to zero and set the standard diviation to 0.01 to 0.1. Also for all the noise, I add them indepentently. After adding the noise, I maintain the weights for each particles, that is to set $\alpha_{t+1|t}^{(k)} = \bar{\alpha}_{t|t}^{(k)}$
2) **Update** After observing the lidar data, I transform the lidar to the head frame and then to the body frame. After that I use the pose of each particles to calculate the positions of the lidar scan in the World frame. The detail of transformation is discussed in the transformation section. After that, I use the scan and map to calculate the correlation score by using the following formula.

$$corr(z, m, x) = \max_{\Delta x \in D} count(\text{Body2World } (z, x + \Delta x), m) \tag{2}$$

$$p_h(z|x, m) = \frac{e^{corr(z,m,x)}}{\sum_v e^{corr(v,m,x)}} \tag{3}$$

$$\tag{4}$$

Here the count functon would count the number of points that agree with each other. Since $p_h(z|x, m) \propto corr(z, m, x)$, we use $corr(z, m, x)$ to update the weights and then normalize weiths to sum to 1. Due to the neumerical concern, I subtract the largest value of all the $corr(z, m, x)$ from each $corr(z, m, x)$. To be more specific, I use:

$$\begin{aligned} corr(z, m, \mu_{t+1|t}^{(k)}) &= corr(z, m, \mu_{t+1|t}^{(k)}) \\ &\quad - \max_{\mu_{t+1|t}^{(k)}} (corr(z, m, \mu_{t+1|t}^{(k)})) \\ \alpha_{t+1|t+1}^{(k)} &= \alpha_{t+1|t}^{(k)} corr(z, m, \mu_{t+1|t}^{(k)}) \\ \alpha_{t+1|t+1}^{(k)} &= \frac{\alpha_{t+1|t+1}^{(k)}}{\sum \alpha_{t+1|t+1}^{(k)}} \end{aligned}$$

I also add a variation in yaw of each particle. For each particle, I would add $\Delta yaw \in \{-0.2, -0.1, 0, 0.1, 0.2\}$ to the original yaw. And calculate the correlation for all of them and choose the one with the largest correlation as the yaw value for this particle so as to make the particles have a better estimation of the position and potentially reduce the needs of large amount of particles to achive the same performance.
3) **Resampling** When the number of effective particles ($N_e ff = \frac{1}{\sum_{k=1}^{N} (\alpha_{t|t}^{(k)})^2}$) is less than a threshold $20\%N$ where N is the total number of particles, I will do a resampling.

C. *Mapping*

I use occupancy grid to do the mapping and assume each cell m_i is independent. Occupancy grid discretize continuous space into a set of cells each with a binary value – one means occupied while zero means free. I maitain the cell log-odds ratio $\lambda_{i,t}$. I choose the particle with the highest weight as the estimation of current state. And transform the lidar to the world frame using this particle. Then I calculate the positions of lidar points in the occupancy grids. Then I use 2D Bresenham's ray tracing algorithm to search for the free cell in the occupancy grids. Decrease the $\lambda i, t$ for the free cells by the predefined log-odds and increase the $\lambda_{i,t}$ for occupied cells by the same log-odds. If the $\lambda_{i,t}$ of the cell is greater than some threshold, then predict the cell as occupied otherwise predict it as free.

$$\lambda i, t = \lambda i, t-1 + \Delta\lambda i, t-1 \tag{5}$$

Here log-odds ratio $\lambda_{i,t} = log(o(m_i|z_{1:t}, x_{1:t}))$ where $o(m_i|z_{1:t}, x_{1:t}) = \frac{p(m_i=1|z_{1:t},x_{1:t})}{p(m_i=-1|z_{1:t},x_{1:t})}$. In this project, we simply assume the $\Delta\lambda i, t = const$. And I choose to set this constant to be $\log(4)$. The threshold of free and occupied cell is also a hyper-parameter. And this one would influence the results of map and reflect the prior belief of the map. Like if the threshold is high, then we tend to predict free cell. Simiplarly, if it is low, then we tend to predict occupied cell. After experiments, I find out 0.6 to 4 works good for this project. Also I do a clip operation on the log-odds ratio to make the ratio whithin some range so that accumulated error can be corrected and the model would not be over confident. Also before doing the prediction and update step, I use the first scan to update the map as the initial map.

D. Transformation

In this subsection, I will discuss the transformation and ground removal techniques used in both SLAM and texture map.

1) **Lidar To Head frame** First transform the lidar points from polar frame to cartesian coordinate. Use the head angles in the joint angle data to build the rotation matrix. First do a rotation around y-axis with angle `head` and then a rotation around z-axis with angle `neck`.

$$x_{head} = R_z(yaw) * R_y(pitch) * x_{lidar} + (0, 0, 0.15)^T \tag{6}$$

Here the $0.15m$ is the distance between lidar and head.

2) **Head To Body frame** Since the head frame and body frame only consists of a z-axis offset. Thus I can use equation.7

$$x_{body} = x_{head} + (0, 0, 0.33)^T \tag{7}$$

Here 0.33 is the distance between head and center of mass.

3) **Body To World frame** Take the particle with the highest weights as the estimated state i.e. $state = [x, y, yaw]$. Then do the following transformation.

$$x_{world} = R_z(state[2]) * x_{body} + (state[0], state[1], 0.93)^T \tag{8}$$

4) **World To Occupancy Grid** Use the following equation to calculate the coordinates of the cells.

$$x_{cell} = ceil((x[0] - Map.xmin)/Map.res) - 1 \tag{9}$$
$$y_{cell} = ceil((x[1] - Map.ymin)/Map.res) - 1 \tag{10}$$

Here $Map.res$ is the resolution for the occupancy grid that is the length of each cell. $Map.xmin/Map.ymin$ and $Map.xmax/Map.ymax$ is the min and max value for the occupancy grid in meters. This can be configured in the configuration file `config.yaml`. For lidar0, I choose to set $res = 0.05m$, and $-20m$ for $Map.xmin/Map.ymin$ and $20m$ for $Map.xmax/Map.ymax$.

5) **depth image To 3D image frame** For the depth image, we have the information of Z, making it possible to project the image back to the 3D space.

$$P_o = \begin{bmatrix} X_0 \\ Y_o \\ Z_o \end{bmatrix} = Z_o K^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{11}$$
$$P_c = R_r^{-1} P_o \tag{12}$$

where K is the intrinsic matrix. P_o and P_c is the representation in optical frame and camera frame. R_r is the rotation from a regular to an optical frame.

6) **IR To RGB image frame** Use the extrinsic matrix R and T, we can transform the points from 3D camera frame of depth $c - depth$ to 3D camera frame of rgb $c - rgb$. Use the intrinsic matrix K_{rgb}, we can then project the points in the camera frame to the optical frame and then the pixel frame.

$$P_{c-rgb} = R * P_{c-depth} + T \tag{13}$$
$$P_{o-rgb} = R_r * P_{c-rgb} \tag{14}$$
$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K_{rgb} * P_{o-rgb}/Z_{o-rgb} \tag{15}$$

E. Texture Map

After transfering depth image to 3D point cloud, assign the colors to the 3D points in the depth image frame. Then find the correct trajectory of current time stamp. Transform the 3D points to the head frame, then to body frame by using the head angles at current time stamp and finally to the world frame using the current pose. Only keep the points that have the z-axis less thatn $0.1m$ – keep the floor 3D points. Then transform those 3D points to the occupancy grid and assign the color to the cell.

1) **Data Preprocessing** Since the camera is distored, thus I use `cv2.undistort` to do the calibration for the depth and rgb image.
Due to the noise of stereo cameras, I choose to clip the depth of the image to $0.4m$ to $4.5m$.

2) **Assign Color to the points in the depth 3D image frame** Since we have the depth image, we can projec the points back the 3D image frame. And use the extrinsic matrix between IR and RGB to calculate the points in the RGB image frame. And use RGB's intrinsic to calculate its pixel representation. Then we can assign the colors to the 3D points in the depth image frame.

3) **Associate the trajectory, head angles and rgb/depth data** While doing SLAM ,I choose to record the trajectory, head angles and the time stamp so that the texture map would be much faster without the need to

do the whole SLAm again! After that use the same synchronization technique discussed in the data processing section to find the data in the current time frame.

4) **Transform from depth image frame to the grid cells** Use the techniques discussed in the transformation section to transfrom the depth image to the points cloud in the 3D image frame. Then use the distance of kinect and head ($0.07m$) to transform the points to the head frame. Then transform points to body frame and then use the current pose to transform points to the world frame. And transform them to the occupancy grids. Assign the color to the corresponding cells.

IV. Result

In this section, I will show the results of the experiments and analyze the results and show the techniques that I have used to further improve the performance.

A. *Mapping using the first scan*

The result for the first scan of lidar0 dataset is shown in the figure.1. The black cell is predicted as free cell while the black one is occupied. We can see a rough shape of the wall and corridor. The shape is not very solid since the lidar scan is very sparse. This result can be used to verify the correctness of transformation by looking at the map like figure .2 generated in the future time stamps.

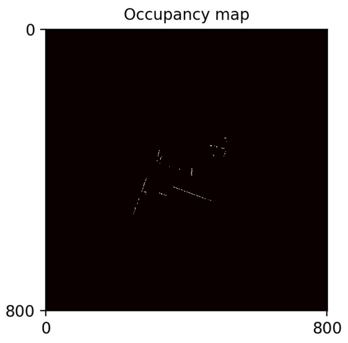


Fig. 1: Mapping result of first lidar scan.

B. *Dead-Reckoning*

Here I will present the result for dead-reckoing – use prediction-only particle filter with no noise and single particle. The results of mapping and trajectory prediction is shown in figure.2. As we can see, the wall starts to shift and the fringe of the wall is unclear. Also there are some sparse points outside of the room. This is due to the result of drift and error of motion model. Also the trajectory seems to oscillate. And the start point and end point is very close! Thus the prediction step is correct.

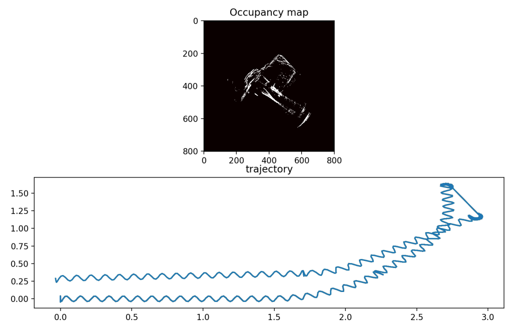


Fig. 2: Results for dead-reckoing on dataset 0. The above figure is the occupancy grid and the below plot is the trajectory.

C. *SLAM results*

I have reported the generated Gif file for better vitualizing the process of SLAM. Please see the file `lidar0.gif`, `lidar3.gif` and `colormap.gif`. Those files are in the zip file that I submitted in the code section. Also, I have attached all the images for vitualization and analysis at the end of the report. The images of generating map and trajectory at different time stamps are shown at the end of the report as well as the textured map.

The best result I can get for Lidar0 is shown in figure.3. But the parameters for this result can not generalize well to the othe dataset. So in the following part, I choose to report the parameters that could have better generalization and show the corresponding result.

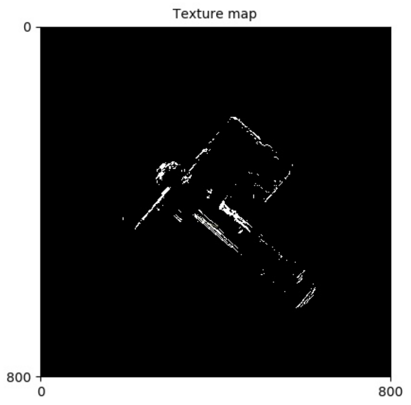


Fig. 3: Results for lidar dataset 0

The result for lidar0 dataset is shown in the figure.4. Here I choose the $\Delta\lambda_{i,t} = log(4)$ I clip the log-odds ratio to avoid too large or too small values to force the value in the range $-10log(4)$ to $4log(4)$. And the threshold for deciding the free/occupied is $2log(4)$ – if cell's log-odds ratio is greater than this threshold, then it is predicted as occupied otherwise as free. Also I experimented with the number of particles, $10, 20, 100, 1000, 2000$. And I notice that as the number of particles increase, the results tends to be better since we have a larger probability to find a better pose estimation. Also for a better calculation of the correlation function, I add the Δyaw in the update step. In the prediction step, I choose to add noise proportional to the magnitude of the motion input as discussed in the technical approach – set the standard diviation of noise to 10% of the relative delta odometry input and zero mean. Also for the numerical stability of softmax operation, I choose to normalize the input of softmax to be less than zero by subtracting the maximum value.

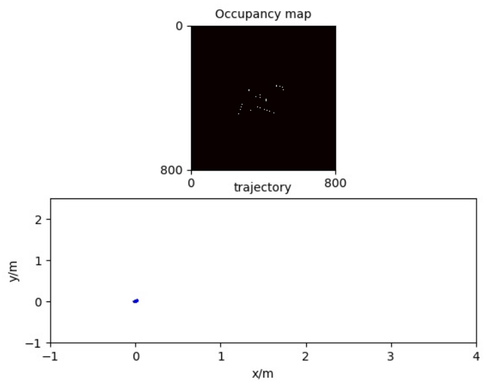
Robot will first go staight and then turn around before moving back. As robot goes straight, the map is easier to grow. But at the turning point, due to the larger yaw value, it is harder to estimate the pose. During experiments, I find that if the number of particles or the noise is too small like $1e-5$ is very small, the wall will shift a lot at the turning point. The result will just like the dead-reckoing in figure.2. If we add more particles like $50-1000$, then the map will become more stable. This is because we have more particles to estimate the true pose of the robot. Also the yaw variation is very important. I add the variation of yaw in the map correlation step, and find that after doing this, I can reduce the number of particles from 200 to 50 to reach the same performance. **Some improvements I have tried and discussion**

1) Adding Δyaw value to the correlation calculation. Add the Δyaw value with the highest correlation value to the yaw value of the particles. In the project, I set $\Delta yaw \in \{-0.2, -0.1, 0, 0.1, 0.2\}$. After doing this, I can make adjustment of particle's position according to the different oritation of the robot so as to make robot have better estimation of the position which could agree better with the current map.
2) Magnitude of noise. In motion model, I will add Gaussian noise with zero mean. In order to make the noise whithin reasonable range, I choose to add noise proportional to the motion input. For example if the motion input is x, y, yaw, then I will set the standard diviation of the noise for x as $\frac{|x|}{10}$, noise for y as $\frac{|y|}{10}$ and noise for yaw as $|yaw|$. The magnitude of the noise if very important in the experiment, if the noise is too high, then the map will shift a lot. If it is too small we can't get the proper estimation of the true state. Also, I add more noise to the yaw value because in the dead-reckoing, the map become bad just when the robot is turning around. Thus I choose to encourage particles to explore more on the yaw axis.
3) Comparison with dead-reckoing. As we can see the results of full SLAM is much better than the dead reckoing. This is because dad-reckoing simply add the odometry to previous state, thus the error will accumulate and finally lead to inaccurate estimation. After adding noise, update and resampling step, we can have many particles and we can use the correlation and resampling to filter out impossible positions, leading to better approximations.
4) Analysis of loop closure. The start point and end point of trajectory in lidar0 dataset works pretty good, indicating that the algorithm can find the correct position and adjust the position accordingly.
5) Comparison between different dataset. As I have mentioned, the best parameters for lidar0 dataset works not very well in the other dataset. As shown in the figure.4 and figure.5, the wall of lidar0 is more clear while the war of lidar3 is more vague. Thus the parameters works better in the lidar0 dataset. This is due to the scene of lidar0 is simpler while lidar3 has slightly convoled scene like the asphalt door and some metal materials in the door which could potentially influence lidar's intensity.
6) Hardness of finding one set of parameters to generate to all the dataset. For example, I use the exact same parameters and apply the algorithm to the lidar1 dataset. The result is shown in the figure.6. As we can see, when the robot moves forward, the result is pretty good. But when the robot starts the second turning, the results start to get poor. This is due to that ou previous parameters wokrs good for simpler scene. While lidar1 contains longer distance and harder scenes, making the old parameters hard to generalize to this scene.

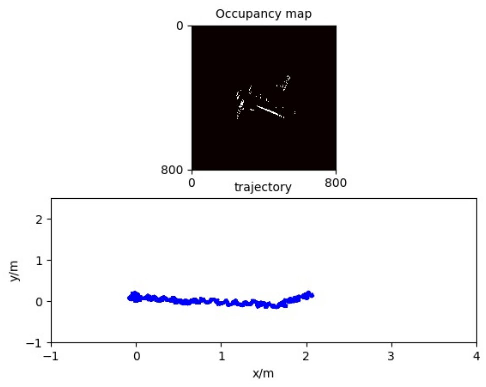
The results for the lidar3 dataset is shown in the figure.5.

D. Textured Map

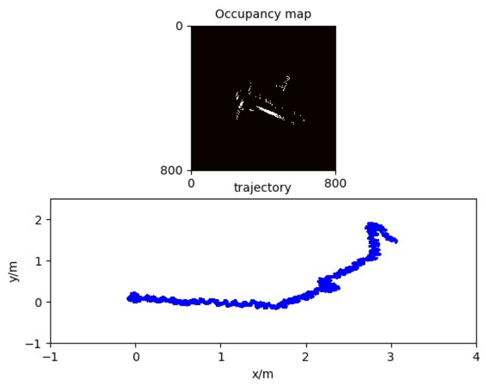
The result for textured map for lidar dataset 0 is shown in figure.7. Due toe the limited accuracy of depth camera and depth camera cannot accuratly estimate the depth in too short or too long distance, thus I choose to clip the depth so the depth is within the range $0.5m$ to $6m$. After assigning the



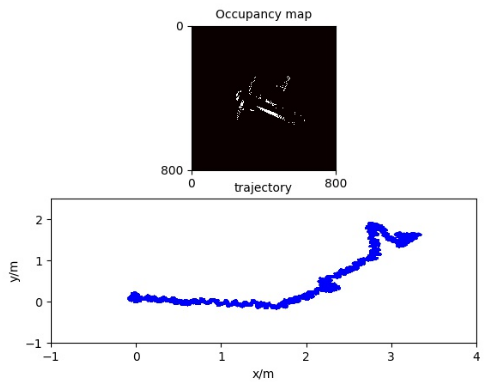
(a) time stamp 1



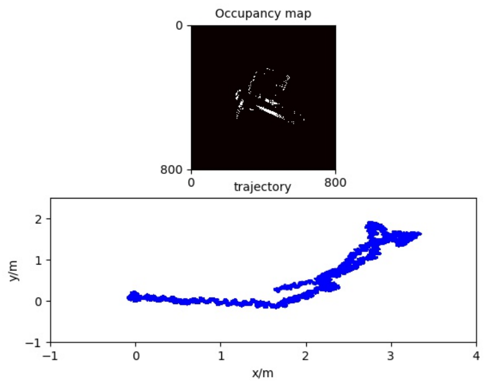
(b) time stamp 4000



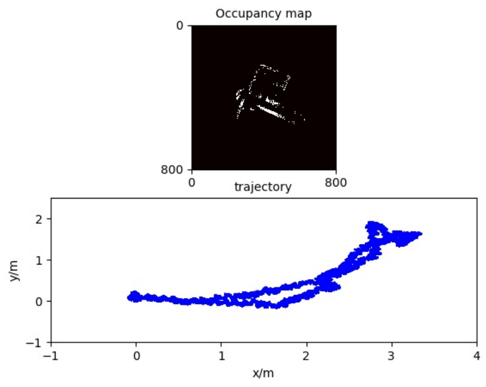
(c) time stamp 6800



(d) time stamp 9010



(e) time stamp 10510



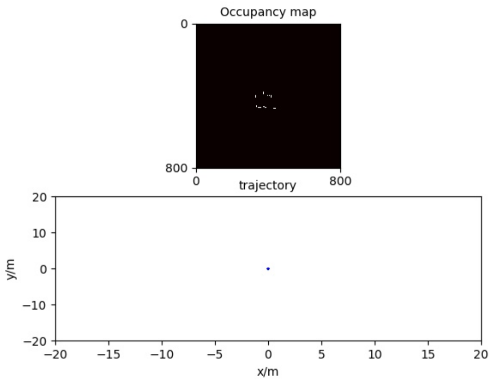
(f) time stamp 12048

Fig. 4: SLAM results for lidar0 dataset. The above figure is the occupancy grids while the below is the trajectory of the robot. Here I plot 6 figures. Each is a result at the time stamp specified.

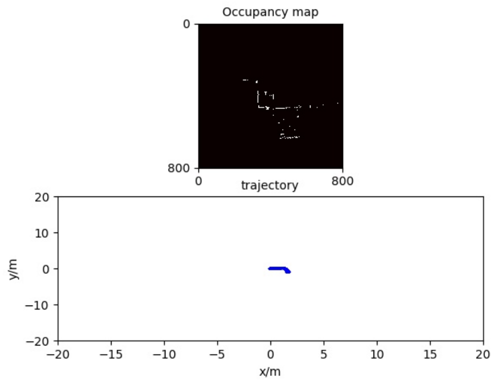
point clouds generated from depth image with the rgb value and transfering them into the world frame, I choose toe keep the points with the z axis less than $0.1m$ as the ground. And then project those points to the cell and color the cell. As figure .7 shows, as robot moves the cells are gradually colored with the color of the floor.

Further improvements Here I brefiely discribe some algorithms that I think could improve the performance.

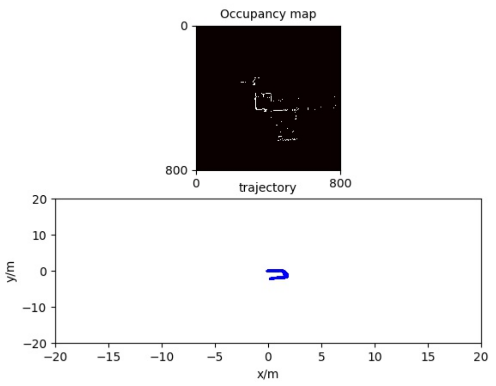
1) Sometimes the robot will color the wall as well. This is because in the original picture the foot of the wall have some grey color. Thus just use the threshold to filter the floor is not accurate enough. To improve this, I think we can utilize the information of the occupancy grid. That is we only color the cells that is free.
2) Use rgb and depth image to help localization. I think we can also use the 3D point clouds generated by



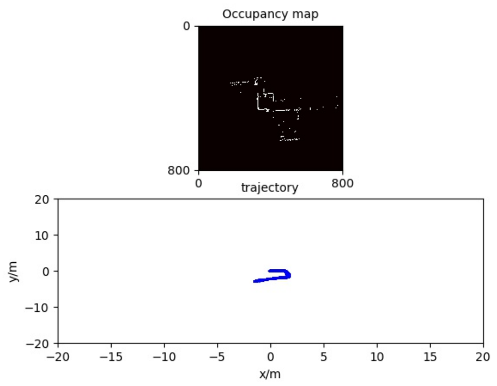
(a) time stamp 1



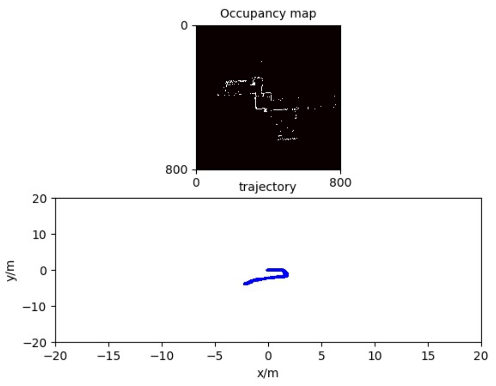
(b) time stamp 4000



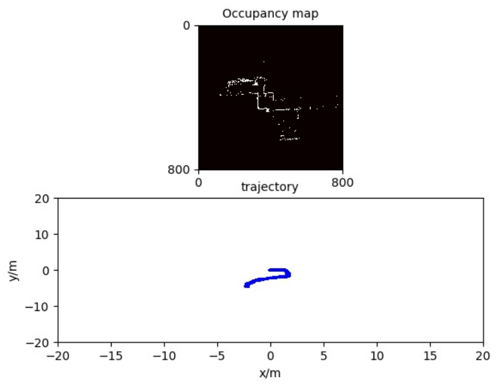
(c) time stamp 6001



(d) time stamp 8001



(e) time stamp 1001



(f) time stamp 12385

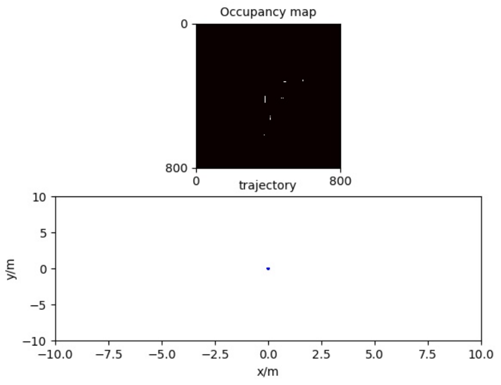
Fig. 5: SLAM results for lidar3 dataset. The above figure is the occupancy grids while the below is the trajectory of the robot. Here I plot 6 figures. Each is a result at the time stamp specified.

depth and rgb image to help do the update step and map correlation calculation. Currently I only use the lidar scan to help localization and after the whole slam process do the coloring. But I think we can also use the rgb and depth image to help the robot localize itself and do the coloring at the same time. In this project, due to the time limit, I choose the former approach with the computation concer. But I think utilize multi-sensor information would also help robot do the slam.

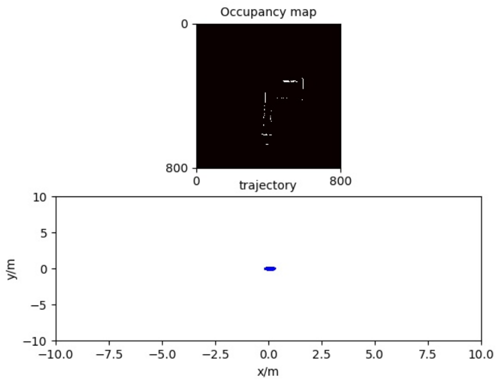
References

[1] Guoquan P Huang, Anastasios I Mourikis, and Stergios I Roumeliotis. Observability-based rules for designing consistent ekf slam estimators. *The International Journal of Robotics Research*, 29(5):502–528, 2010.

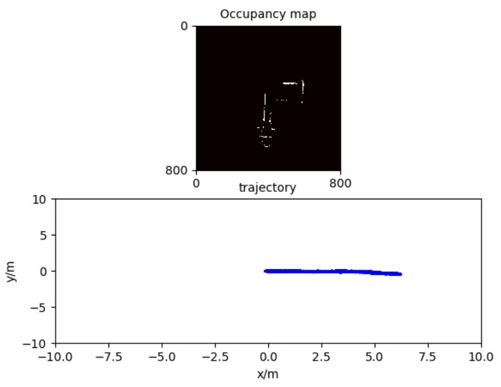
[2] John J Leonard and Hugh F Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on robotics and Automation*, 7(3):376–382, 1991.



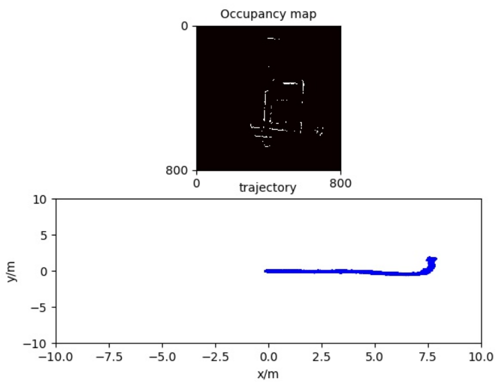
(a) time stamp 1



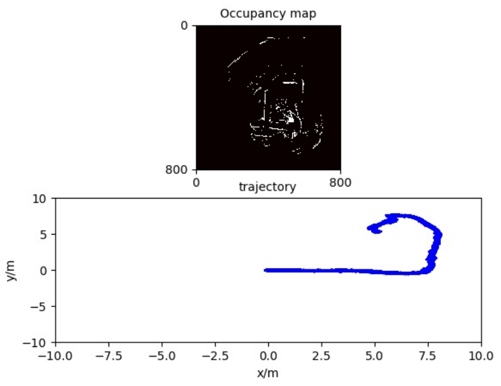
(b) time stamp 2001



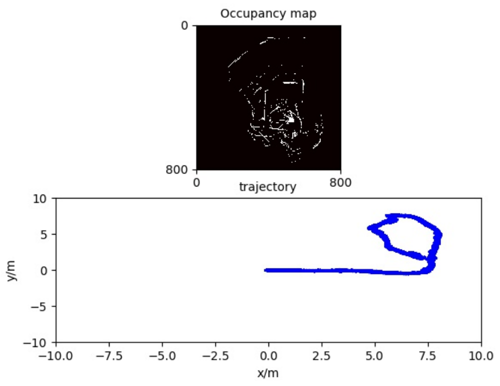
(c) time stamp 10001



(d) time stamp 14001



(e) time stamp 28001

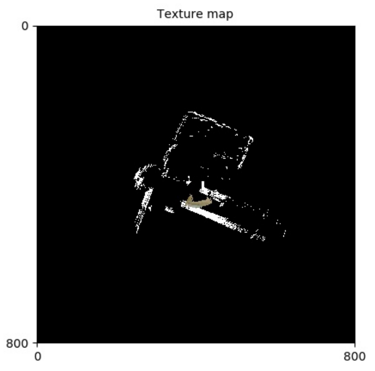


(f) time stamp 34208

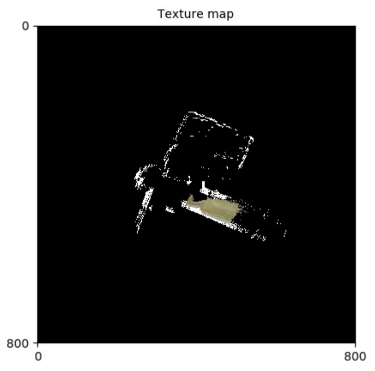
Fig. 6: SLAM results for lidar1 dataset. The above figure is the occupancy grids while the below is the trajectory of the robot. Here I plot 6 figures. Each is a result at the time stamp specified.

[3] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598, 2002.

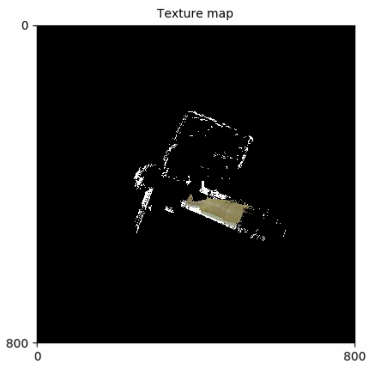
[4] Sebastian Thrun and Michael Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, 25(5-6):403–429, 2006.



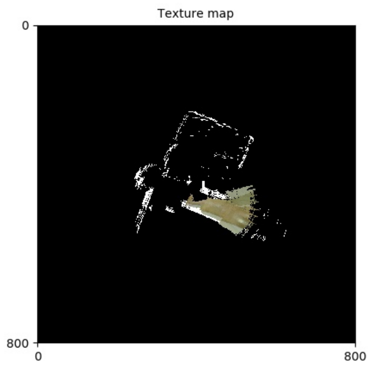
(a) time stamp 60



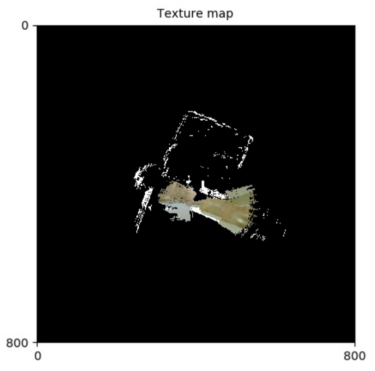
(b) time stamp 80



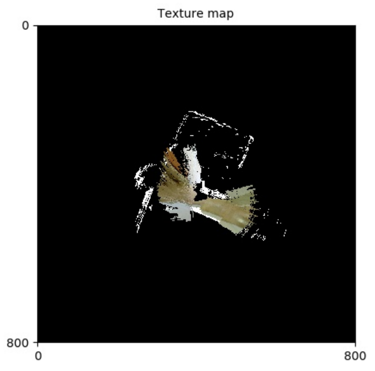
(c) time stamp 100



(d) time stamp 120



(e) time stamp 180



(f) time stamp 227

Fig. 7: Results of textured map. (a)-(f) is the texture map over time. And iteration 227 is the last image and the last time stamp. The trajectory is the same as the lidar0 dataset.